METHOD AND APPARATUS FOR HANDLING TRANSIENT MEMORY ERRORS

Inventors

Philippe Bernadat
10 rue de la Suze
38760 Varces
France

Dejan Milojicic
3989 La Donna Ave.
Palo Alto, CA 94306

Guangrui Fu
3580 South Court
Palo Alto, CA 94306

Alan Messer
1055 Manet Dr. #26
Sunnyvale, CA 94087

Assignee

Hewlett Packard Company

METHOD AND APPARATUS FOR HANDLING TRANSIENT MEMORY ERRORS

FIELD OF THE INVENTION

The present invention generally relates to memory management in data processing systems, and more particularly to handling transient memory errors.

BACKGROUND

Society's demand for high-availability computing systems is growing along with society's dependency on computers for various services. For example, Internet Data Centers (IDC), Internet Service Providers (ISP), or Application Service Providers (ASP) provide the support for many computing needs. To meet the demand in a way that is affordable to users, computing systems are increasingly being built with commodity hardware and software. Unfortunately, reliability is sometimes sacrificed in systems with commodity parts.

For example, commodity memory components are susceptible to soft errors. A soft error is a transient memory error that has been detected by the hardware but not corrected. Many operating systems respond to soft errors by halting and then rebooting. System reboots are costly in terms of lost production time. If the resources of an IDC, ISP, or ASP are unavailable because of a system reboot, customers' needs may be unmet or frustrated. If computing resources are unavailable too often or for too long, customer dissatisfaction and customer defections may result. Thus, while commodity parts address the requirement of affordability, the requirement of high availability may be sacrificed.

A method and apparatus that address the aforementioned problems, as well as other related problems, are therefore desirable.

SUMMARY OF THE INVENTION

The invention provides in various embodiments methods and apparatus for managing memory of a data processing system. In one embodiment, memory objects are allocated in response to memory allocation requests. Each object has an associated plurality of addresses. Type-identifier codes are respectively stored in association with the memory objects. Upon detection of a transient memory error at a memory address a recovery action is selected and performed based on the type-identifier code of the object that is associated with the erring memory address.

Various example embodiments are set forth in the Detailed Description and Claims which follow.

BRIEF DESCRIPTION OF THE DRAWINGS

Various aspects and advantages of the invention will become apparent upon review of the following detailed description and upon reference to the drawings in which:

FIG. 1 is a flowchart of an example process for allocating memory;

FIG. 2 is a block diagram of multiple memory objects established in accordance with one embodiment of the invention; and

FIG. 3 is a flowchart of an example process for handling memory errors.

DETAILED DESCRIPTION

In various embodiments, the invention categorizes memory objects and responds to memory errors by selecting a course of action based on the category of the object in which the memory error occurred. The course of action may include, for example, ignoring the error, signaling the task that was executing when the error occurred, restarting a system

call or I/O operation, reloading program code into a text area, recovering data prior to a read error (e.g., rebuilding a page table entry, reconstructing a linked list, or recopying memory from user space), reloading a page from storage, or refilling a page with zeros if the page hasn't been modified, or as a last resort halting the system if necessary. Thus, a system in which the invention is implemented has numerous recovery actions that are available as alternatives to system halts and reboots, and the actions can be tailored to the context in which the memory error occurred. Such a system is less susceptible to unnecessary halts and reboots.

FIG. 1 is a flowchart of an example process for allocating memory in accordance with one embodiment of the invention. The memory allocation process is implemented by the operating system and is used by the operating system and by application programs to allocate memory for use during program execution.

At step 102 a memory allocation request is received. The request includes a requested quantity of memory. In another embodiment, the request specifies the type of memory object. By reference to the memory address or object type, the memory manager identifies the cluster of memory objects with which the requested memory object is to be associated. At step 104, the process determines the type of object to which the memory is to be allocated. For example, in one embodiment, the objects are categorized into operating system objects, user application objects, and objects beyond the control of the operating system.

Operating system objects include, for example, task descriptors, I/O buffers, file handles, the kernel stack and other objects. An application object is any memory object allocated to and accessed by an application program (a program other than the operating system). Those memory objects beyond control of the operating system are those that are manipulated by system firmware, for example objects accessed by a BIOS. In one

embodiment, an object type is identified by reference to the value of the program counter to which control is to return upon allocation of the object. The value of the program counter uniquely identifies the requester. In another embodiment, the type of the object is passed as a parameter to the allocation process. In yet another embodiment, a shorter object-type identifier is generated using intermediate preprocessor macros. The preprocessor macro converts the program counter value into an index, for example, a byte-sized integer. The index is used to store the real program counter value in an intermediate table, and used thereafter for reading the program counter value from the table.

The granularity with which objects are categorized refers to the degree to which the system can differentiate between memory errors in different locations. For example, depending on system requirements, the granularity by which operating system objects are categorized can be relatively fine. A fine granularity makes easier the task of selecting a suitable course of action. That is, with smaller objects a wider variety of error responses can be programmed with knowledge of the uses of the objects. For objects belonging to application programs, the granularity can be coarse since the application will either be terminated or signaled if the application is programmed to handle a soft error.

At step 106, the type of the memory object is associated with the object. The association is established so that when a soft memory error is detected, the type of the object in which the memory error occurred can be determined and appropriate action taken. In one embodiment, a type code that describes the object type is stored at a fixed offset within the object itself. A pointer to the allocated memory object is then returned to the requester.

FIG. 2 is a block diagram of multiple memory objects established in accordance with one embodiment of the invention. Each of blocks 152, 154, and 156 represents a memory object. It will be appreciated that FIG. 2 is limited to three objects for illustrative

purposes only. Within each memory block an associated type code is stored. The type code, for example, a program counter value, indicates the type of object to which the memory is allocated.

In an alternative embodiment, if multiple objects of the same type are clustered in a selected area of memory, the object type code is stored in a header that is used to manage the objects in the cluster. As between clusters of different types of objects, the object type codes are stored in the headers at the same offset for ease of reference in determining the type of object in responding to a memory error. Because in some applications objects are infrequently allocated, different types of objects may be clustered in a selected memory area to reduce the overhead associated with maintaining a cluster for each type of object. For clusters having mixed object types, the object type codes are stored in the objects as shown in FIG. 2.

Within object 152, offending location 160 represents an example address at which a soft error is detected. When a soft error is detected, the object to which the offending address belongs is determined, and then the type of the object is determined from the associated type code. Based on the object type, the operating system decides on a suitable course of action. For example, if the object belongs to an application program, the operating system either signals the application that a soft error was detected or ends the application.

FIG. 3 is a flowchart of a process for handling memory errors in accordance with one embodiment of the invention. A memory-error exception handler is invoked when the hardware detects a soft error. At step 202, the exception handler begins the process with an input memory address of the location at which the error was detected.

At step 204, the exception handler looks up the type of object to which the input address belongs. In one embodiment, the operating system groups into clusters objects of

the same type and same size. Thus, to identify the type of the object in which the offending address resides, the exception handler first finds the cluster base address that is nearest and less than the offending address. From the cluster base address, the object is determined (each object in a cluster is of the same size). Once the object is identified, the type code is read from the predetermined location in the object.

At step 206, a recovery action is selected and performed based on the type code read from the object. For example, in one embodiment the recovery actions include ignoring the error, signaling the application, halting the system or fixing the error. To fix an error, for example, a write operation is retried in response to a write error. If an error occurs in a program text segment, the operating system reloads the page associated with the offending address. For an I/O buffer, the operating system may re-execute the I/O operation. If the offending address is associated with an object wherein the correctness of the data does not affect semantics of the operating system, the error is logged.

In addition to the example embodiments described above, other aspects and embodiments of the present invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and illustrated embodiments be considered as examples only, with a true scope and spirit of the invention being indicated by the following claims.